Aix*Marseille université
Initiative d'excellence
Année 2019-20

Domaine Sciences et Technologies
Portail René Descartes

Programmation 1 : TP 6
Code UE : S14IN2I1

Projet Astéroïde : partie 1

Les prochains TP sont consacrés au développement d'un jeu en temps réel. Vous travaillerez à partir d'une version inaboutie du jeu, pour y ajouter des fonctionnalités. La version initiale se trouve sur Ametice.

Récupérer l'archive contenant les fichiers du projet sur Ametice et réaliser son extraction dans un nouveau projet.

Attention, votre projet doit être bien organisé : le répertoire `src` contenant les packages, chacun contenant des fichiers sources Java.

Configurer JavaFX, comme pour le TP précédent : import de la librairie et ajout d'options à la machine virtuelle pour le `main`, qui est situé dans la classe `Main` du package `views`.

Vérifier que le programme s'exécute correctement.

Le projet est constituée de nombreuses classes. Vous aurez à modifier les classes des packages `game` principalement, ainsi que `viewModel` et `view` parfois. Vous ne devez pas touchez aux fichiers des autres packages, ni aux classes dont un commentaire indique qu'il ne faut pas les modifier.

1 Modifier les paramètres du programme

1. Le programme est partiellement documenté, via des commentaires au format `javadoc`. Générez la documentation, en suivant ces instructions :
   (a) Créer un répertoire `doc` dans le répertoire du projet.
   Vous pouvez le faire depuis IntelliJ avec un clic droit sur la racine du projet.
   (b) Ouvrir le menu `Tools > Generate Javadoc...`
   (c) Positionner le champ `Output directory` sur le répertoire `doc` que vous venez de créer.
   (d) Ajouter `-html5` dans le champ `Other command line arguments`.
   (e) Valider.
   Si tout se passe bien, le répertoire `doc` se remplit de fichiers, et un navigateur s'ouvre avec la documentation du projet. Vous pouvez aussi accéder au projet en ouvrant le fichier `index.html` du répertoire `doc` dans un navigateur.
2. Lire le source de la classe `Space`.
3. Trouver comment modifier le nombre d'astéroïdes, pour le faire passer à 10. Et comment changer la taille des astéroïdes ?
4. Lire le source de la classe `Asteroid`.

2 Déplacement du vaisseau spatial

Le vaisseau spatial se contrôle avec les touches flèches du clavier. Dans cette section, on se concentre sur la classe `Spaceship`.

1. Que se passe-t-il actuellement lorsqu'on tente de déplacer le vaisseau spatial ?

2. On souhaite suivre les lois physiques un peu plus fidèlement. On s'intéresse dans un premier temps uniquement au déplacement en ligne droite. D'après nos connaissances en physique, le déplacement d'un objet ponctuel est régi par sa masse m et la somme des forces s'y appliquant $\vec{F}$. Si on note $\vec{p}$, $\vec{v}$ et $\vec{a}$ les vecteurs position, vitesse (la *vélocité*) et accélération respectivement, on a :

$$\begin{array}{lccc} & m\vec{a} = \vec{F}, & \vec{a} = \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}, & \vec{v} = \frac{\mathrm{d}\vec{p}}{\mathrm{d}t} \\ \text{c'est-à-dire} & m\vec{a} = \vec{F}, & \mathrm{d}\vec{v} = \mathrm{d}t \times \vec{a}, & \mathrm{d}\vec{p} = \mathrm{d}t \times \vec{v} \end{array}$$

Dans ces notations, $\mathrm{d}t$ est la différentielle de temps, $\mathrm{d}\vec{p}$ la différentielle de position et $\mathrm{d}\vec{v}$ la différentielle de vitesse. Ce qui veut dire que lorsque $\mathrm{d}t$ secondes passent (pour $\mathrm{d}t$ suffisamment petit),
— le temps passe de t à $t + \mathrm{d}t$,
— la position du vaisseau passe de $\vec{p}$ à $\vec{p} + \mathrm{d}\vec{p} = \vec{p} + \mathrm{d}t \times \vec{v}$,
— la vélocité du vaisseau passe de $\vec{v}$ à $\vec{v} + \mathrm{d}\vec{v} = \vec{v} + \mathrm{d}t \times \vec{a}$.
Comme dans les TP précédents, Les équations physiques nous donnent donc les instructions nécessaires pour mettre à jour, à chaque pas de temps, les caractéristiques déterminant l'état du vaisseau.

> Actuellement, le vaisseau n'a qu'une seule caractéristique représentée dans la classe `Spaceship` : sa position. Ajouter une propriété pour sa vélocité $\vec{v}$.

3. S'assurer que la vélocité est initialement nulle.
4. Le vaisseau est soumis à une accéleration nulle par défaut, ou par l'accélération constante procurée par son moteur lorsque celui-ci est actif.

> Ajouter une propriété contenant un nombre correspondant à la puissance $|\vec{F}/m|$ du moteur :

```
private static final double MAIN_ENGINE_POWER = 50;
```

5. L'accéleration $\vec{a}$ du moteur allumé est donc donnée par `direction.multiply(MAIN_ENGINE_POWER);`.

> Écrire une méthode `public Vector getAcceleration()` qui retourne le vecteur d'accélération du vaisseau, selon l'état du moteur (allumé ou éteint).

6. Modifier la fonction `update`, en utilisant les lois physiques décrites ci-dessus, pour que la trajectoire (la position et la vitesse) du vaisseau soit correctement calculée.

Utilisez les méthodes des objets de la classe `Vector` pour réaliser les opérations vectorielles. Vous pouvez aussi vous inspirer de la méthode `Asteroid.update`.

7. Tester le comportement du vaisseau maintenant. En quoi diffère-t-il du comportement initial ?
8. Modifier la valeur du paramètre `MAIN_ENGINE_POWER` pour obtenir une accélération qui vous convienne : ni trop faible, ni trop forte.
9. On s'intéresse maintenant à faire tourner le vaisseau. Pour cela, le vaisseau dispose de moteurs latéraux qui, lorsqu'ils sont allumés, provoquent une rotation du vaisseau sur lui-même autour de son centre, soit dans le sens horaire, soit dans le sens anti-horaire.

> Dans un premier temps, ajouter des propriétés au vaisseau, déterminant si les moteurs latéraux gauches ou droits sont allumés.

10. Ajouter des méthodes permettant d'allumer ou d'éteindre les moteurs latéraux.
11. Lorsqu'un moteur latéral est allumé, le vaisseau tourne sur lui-même au rythme d'un tour complet par seconde (sa vitesse angulaire), dans un sens ou dans l'autre.

Écrire une méthode `updateDirection`, paramétrée par le délai `dt`, modifiant le vecteur direction par une rotation d'un angle proportionnel au temps écoulé.

12. Faire en sorte que lorsque la position du vaisseau est mise à jour, sa direction soit elle aussi mise à jour.
13. Ouvrir la classe `ViewModel` du package `viewModel`. Cette classe assure la communication entre la fenêtre graphique et le modèle de notre jeu.

    En vous inspirant des méthodes présentes, ajouter dans cette classe des méthodes pour démarrez et stoppez les moteurs latéraux du vaisseau du jeu.

14. Dans la classe `View` du package `view`, repérez les deux méthodes `handleKeyPressed` et `handleKeyReleased`. Ces deux méthodes choisissent les actions à effectuer en fonction de la touche pressée ou relachée par le joueur. `handleKeyPressed` est partiellement complétée en commentaire. La structure de contrôle `switch` permet de choisir une action selon la nature de la touche pressée (`UP`, `LEFT`,...).

    Modifier les cas `LEFT` et `RIGHT` en choisissant dans chaque cas l'action sur `viewModel` correspondant à la touche pressée. Faire de même pour `handleKeyReleased`.

15. Lancer le jeu et vérifier votre travail.
    Le vaisseau tourne-t-il correctement ? Au besoin, faites les corrections nécessaires.
16. Modifier la vitesse angulaire pour obtenir une vitesse de rotation qui vous convienne.
17. Comme pour l'accélération du moteur, il est bon d'avoir une constante définissant cette vitesse angulaire en dehors des méthodes de calcul. Créer donc cette propriété.
18. Ajouter une action supplémentaire, permettant de freiner le vaisseau.
    Le freinage fonctionne comme l'accélération, mais en sens inverse et avec une acceleration plus faible. Ainsi, il faut modifier le calcul de l'accélération, ajouter des propriétés représentant l'état du moteur de recul, ajouter des méthodes pour accéder ou modifier l'état du moteur de recul, puis liés ces méthodes à la touche `DOWN` (en créant les méthodes nécessaire dans `ViewModel`). La puissance du moteur de recul devra être déterminée par une constante de la classe `Spaceship`.

3 Consommation

Les moteurs consomment de l'énergie. Ajoutez des propriétés au vaisseau pour l'état de son stock d'énergie. Modifiez les méthodes de sorte que le vaisseau consomme de l'énergie selon les actions effectuées. Comme il fonctionne à l'énergie solaire, l'énergie disponible augmente lentement avec le temps (mais pas assez pour que les moteurs restent allumés en permanence), sans dépasser les capacités de stockage du vaisseau. En absence d'énergie, les moteurs ne s'allument plus.

On se place dans la classe `Spaceship`.

1. Ajouter une propriété `fuel` indiquant la quantité de carburant dans le réservoir. On mesurera cette quantité en secondes de consommation du moteur principal.
2. Ajouter une constante définissant la capacité du réservoir (par exemple 5 secondes de consommation).
3. Ajouter une constante définissant la quantité de carburant rechargée par seconde (par exemple 0.2 secondes de consommation par seconde). Puis des constantes définissant la consommation des différents moteurs (1 pour le moteur principal, 0.3 pour les moteurs latéraux, 0.5 pour le moteur de recul).

4. Ajouter une méthode privée `getCurrentConsumption()` retournant la consommation net par seconde du vaisseau.
   On prend en compte l'état de chaque moteur, ainsi que le taux de recharge du carburant.
5. Ajouter une méthode privée `getAutonomy(double dt)`, calculant combien la réserve de carburant du vaisseau peut tenir dans l'état actuel des moteurs. Si ce temps excède `dt`, on retourne `dt`.
   Par exemple, si les moteurs principaux et latéral gauche sont allumés, le vaisseau consomme $1+0.3-0.2$ net par seconde, et épuise sa réserve actuelle r de carburant en $t = r/(1+0.3+0.2)$ secondes. On retourne le minimum de t et dt. Si tous les moteurs sont éteints, le vaisseau produit 0.2 par seconde jusqu'à remplissage du réservoir, et donc a une réserve illimitée à ce rythme, on retourne alors `dt`.
6. Modifier la méthode `updateVelocity`, en prenant en compte le temps d'allumage des moteurs calculé par `getAutonomy`.
   L'accélération doit être multiplié non pas par `dt` comme avant, mais par l'autonomie au régime actuel. Ainsi, s'il n'y a pas assez de carburant pour tenir les moteurs allumés pendant un temps `dt`, l'accélération sera d'autant moins forte.
7. Modifier de façon similaire la méthode `updateDirection`.
   La vitesse de rotation est ajustée selon l'autonomie.
8. Dans la méthode `update`, ajouter après la mise-à-jour de la vélocité, de la direction et de la position, une mise-à-jour du niveau de carburant.
   Utiliser de nouveau l'autonomie et la consommation actuelle. Assurez-vous que le niveau de carburant est positif et inférieur à la capacité du réservoir.
9. Ajouter une méthode publique retournant le pourcentage de carburant dans le réservoir.
10. Dans la classe `ViewModel`, ajouter aussi une méthode fournissant le pourcentage de carburant dans le réservoir du vaisseau spatial.
11. Dans la classe `CanvasView`, ajouter une méthode `renderFuel` prenant le pourcentage de carburant du vaisseau, et l'affichant sur le canvas sous la forme d'une barre rectangulaire sur un côté de l'écran.
    La longueur de la barre est proportionnelle au taux de remplissage du réservoir : la barre doit faire la longueur ou la hauteur de l'écran quand le réservoir est plein (avec si vous le souhaitez une petite marge), et disparaitre lorsqu'il est vide. Voici quelques informations pour vour aider :
    — La propriété `context` gère le dessin : pinceau et feuille.
    — sa méthode `setFill` permet de changer la couleur de remplissage. Les couleurs sont `Color.WHITE`, `Color.BLUE`, ... utilisez l'auto-complétion.
    — sa méthode `fillRect(x,y,width,height)` permet de peindre un rectangle.
    — le canvas fait 800x800 pixels, l'origine est en haut à gauche.
12. Dans cette même classe, modifier `render()` pour ajouter le rendu du niveaude carburant.
    Vérifier que tout fonctionne comme prévu.

4 Animation des moteurs

Ce serait mieux si on visualisait quand les moteurs sont allumés. Pour cela, il faut modifier l'affichage du vaisseau. Le répertoire `resources` contient les images nécessaires pour cela.

1. Dans la classe `ViewModel`, ajouter des méthodes permettant de savoir pour chaque moteur du vaisseau s'il est allumé.
2. Dans la classe `CanvasView`, dans la méthode `render(Spaceship)`, ajouter des instructions, avec des conditions appropriées, permettant d'afficher les images des différents moteurs en action.
   Les images des différents moteurs sont déjà définies par des chaînes de caractères en fin de fichier.
3. Vérifier le bon fonctionnement de cet ajout.